[course name - Algorithm & lab]

1st cls — {Algorithm Design and analysis}

Introduction to Algorithm

Book name → Introduction to Algorithm by Thomas H. ~~Cor~~ corman.

* What is algorithm?

→ set of steps an instruction for completing a task.

[A sequence of computational steps that transforms some input to output] ~~se~~ (SC).

*Complexity Analysis → time and space.

Insertion sort

Example:-

	0	1	2	3	4	5
ann	2	5	1	4	9	3

here, k = 1 - (n-1) → which will apply loops.

{(k-1), 0} / {1, (k-1)}

Algorithm for Insertion sort (A):-

1. for j = 2 to A.length.
2. key [j]
3. i = j - 1
4. while i > 0 and ~~key~~ A[i] > key.
5. A[i+1] = A[i]
6. i = i - 1
7. A[i+1] = key

— x —

2nd class

2 parts of Algorithm :-

- Design (Devide and conquise)
- Analysis

Selection sort → (Incremental approach)

Devide and conquine (Divide, sort, conquine, combine)

Analysis :-

① Time
② space

Ex :- sum(A, n) :-

1. Sum = 0 ——— 1
2. for i=0 to (n-1) ——— n+1
3. Sum = sum + A[i] ——— n
4. Return sum. ——— 1

Time ⟶ 2n + 3

complexity : $O(n)$

space :-

A → n
n → 1
sum → 1
1 → 1

n + 3

space : $O(n)$

Ex: Sum(A, B, n)

1. for i=0 to (n-1) ——→ n+1
2. for k=0 to (n-1) ——→ n
3. C[i][k] = A[i][k] + B[i][k] ——→ n(n+1)
4. Return ——→ 1

Time complexity :- $n+1+n^2+n+n^2+n+1$

$T(n) = 2n^2+3n+2$

T.C :- $O(n^2)$

Space :-

$A \rightarrow n \times n = n^2$

$B \rightarrow n^2$

$C \rightarrow n^2$

$i \rightarrow 1$

$k \rightarrow 1$

$n \rightarrow 1$

$S(n) = 3n^2+3$

S.C :- $O(n^2)$

H.W :- complexity of Binary Search

Ex: Pattern(n):-

1. for i=0 to (n-1) ⟶ n+1
2. for k=0 to (n-1) ⟶ $\frac{n(n-1)}{2}$
3. print: i; ⟶ $\frac{n(n-1)}{2}$
4. print: \n; ⟶ n
5. return ⟶ 1

Time complexity:- $(n+1)+\frac{n^2}{2}-\frac{n}{2}+\frac{n^2-n}{2}+1$

T.C: $O(n^2)$

Ex:-

```
1. for (i=1; i<n; i=i*2)
2.     // code
3. return
```

Time:-

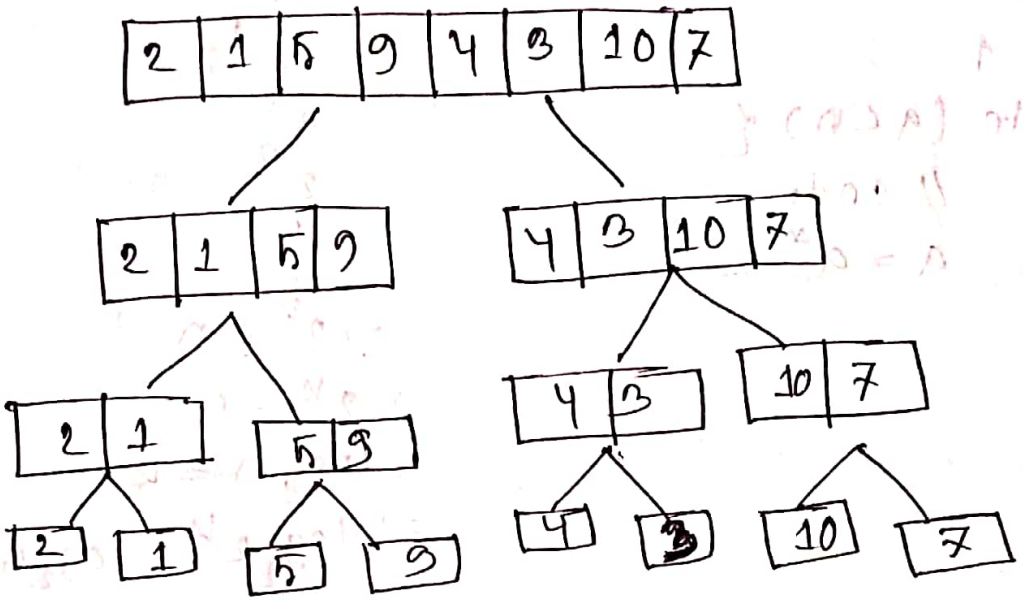
i = 1
i = 2
$i = 2^2$
$i = 2^4$
:
$i = 2^k$

$i > n$
$\Rightarrow 2^k > n$
$\Rightarrow k > \log_2 n$
$\therefore k = \log_2 n$

T.C:- $O(\log_2 n)$

Class - 3

Merge - sort



Algorithm:- (Binary search)

Binary Search (A, beg, end, s-item) :-

1. if beg > end
2. return -1.
3. mid = (beg + end) / 2
4. if A[mid] = s-item
5. return mid
6. if A[mid] > s-item
7. return Binary-search (A, beg, mid, s-item)
8. else
9. return Binary-search (A, mid, end, s-item)

insertion sort, Quick sort [sometimes me Q.S is faster then M.S in some case]

4th class

Time complexity :-

Ex: 1. a = 1
2. while (a < n) {
3. // code
4. a = a * 2
5. }

$a \to 2^0$
$2 \to 2^1$
2^2
2^3
2^k

$\therefore 2^k \leq n$
$\Rightarrow 2^k = n$
~~$\log_2 n = k$~~
$\Rightarrow \log_2 2^k = \log_2 n$
$\Rightarrow k = \log_2 n$

T.C :- $O(\log n)$

Ex:

1. for(i = 1; i < n; i *= 2) {
2. // code
3. }

T.C :- $O(\log n)$

1. i = n
2. while (i >= 1) {
3. // code
4. i = i / 2
5. }

T.C : $O(\log n)$

1. for(i = n; i >= 1; i = i/2)
{
2. // code
3. }

T.C :- $O(\log n)$

$i \to n \to \frac{n}{2} \to \frac{n}{4} \cdots \frac{n}{2^k}$

$\therefore \frac{n}{2^k} \geq 1 \to \frac{n}{2^k} = 1 \to n = 2^k \to k = \log_2 n$

T.C : $O(\log n)$

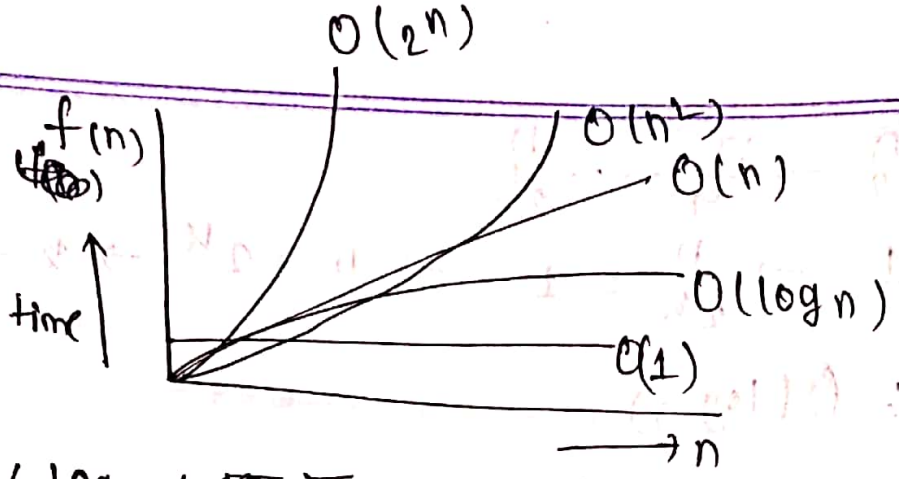
Types of function :-

speed ↓

- $O(1) \longrightarrow f(n) = 20$, area $= \pi r^2$ $\Big\rangle$ $O(1) \longrightarrow$ constant
- $O(\log n) \longrightarrow$ logarithmetic $\to f(n) = 100 \log n$, $f(n) = 10 + 20 \log n$ $\Big\rangle$ $O(n)$
- $O(n) \longrightarrow$ Linear $\longrightarrow f(n) = 2n + 3$, $f(n) = 1000 + \frac{n}{2}$ $\Big\rangle$ $O(n)$
- $O(n^2) \longrightarrow$ Quandic
- $O(n^3) \longrightarrow$ cubic
- $O(2^n) \longrightarrow$ exponential
- $O(n^n) \longrightarrow$ exponential

Growth function :-

n \ f(n)	$\log n$	n	n^2	2^n
1	0	1	1	2
2	1	2	4	4
4	4	4	16	16
8	3	8	64	256
16	4	16	256	65536



$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \dots < 2^n < \dots < n^n$

3 types of case :-

1. Best case
2. Average case
3. Worst case

Assymptonic notation :-

3 types of notation

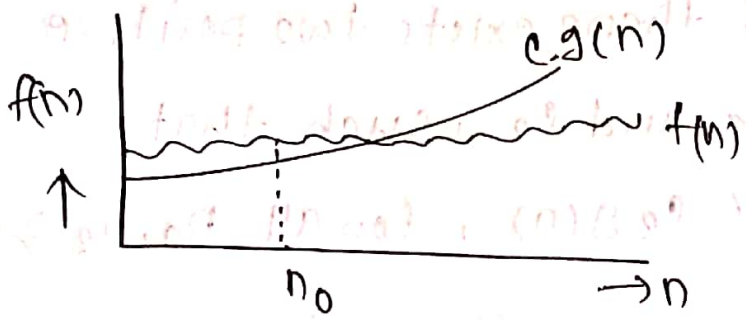
(i) Big. oh (O) → upper bound of a function.

(ii) Big. omega (Ω) → lower bound of a function.

(iii) Big. theta (Θ) → best one → Average bound of a function.

Big O (upper bound) :-

Big-oh notation :- Function $f(n) = O(g(n))$ if and only if there exist a positive constant c such that $f(n) \leq c.g(n)$, for all $n > n_0$.



ex :- $f(n) = 3n + 5$

$\rightarrow 3n + 5 \leq 10n \;,\; n \geq 1$

$f(n) \quad c.g(n) \quad n_0$

$\rightarrow 3n + 5 \leq 8n \;,\; n \geq 1$

$f(n) = O(n)$

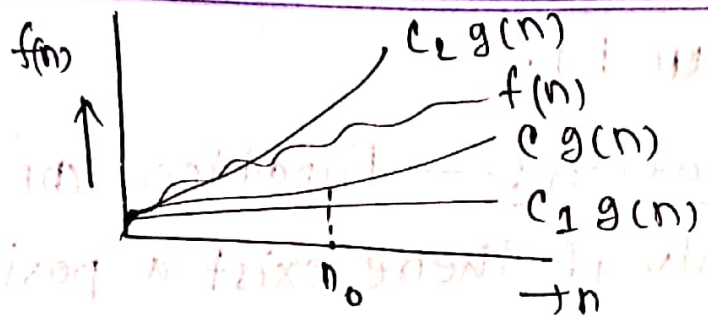
Big-omega notation :- A function $f(n) = \Omega(g(n))$ if and only if there exists a positive constant such that,

$f(n) \geq c\,g(n)$, for all $n \geq n_0$

Ex :- $3n + 5$

$3n + 5 \geq 7n \;,\; n \geq 1$

$f(n) = \Omega(n)$



Big-theta notation:- A function $f(n) = \Theta(g(n))$ if and only if there exists two positive constants c_1 and c_2, such that;

$c_1 g(n) \leq f(n) \leq c_2 g(n)$, for all $n_1, n_2 \geq n_0$

$f(n) = 3n + 5$

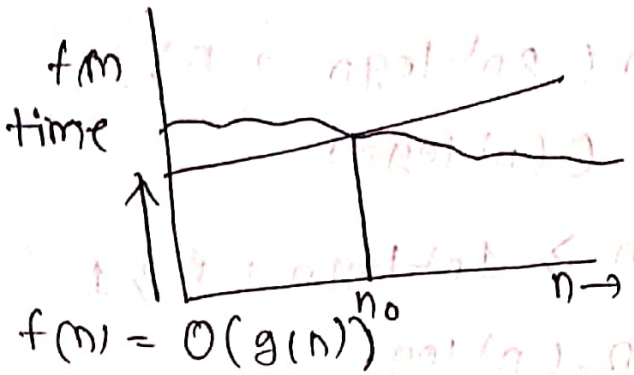
$5n \leq 3n + 5 \leq 8n$, $n \geq 1$

$5n \downarrow c_1$; $3n + 5 \downarrow f(n)$; $8n \downarrow c_2$

$\therefore f(n) = \Theta(n)$

5th class

Asyptotic notation :-



if $f(n) = O(g(n))$

$f(n) = \Omega(g(n))$

then,

$f(n) = \Theta(g(n))$

① $f(n) = 2n^2 + 3n + 4$

② $f(n) = n^2 \log n + n$

$2n^2 + 3n + 4 \longrightarrow \Theta(n^2)$

$n^2 \log n + n \longrightarrow n^2$

① $2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2$

$\Rightarrow 2n^2 + 3n + 4 \leq 9n^2 \; , \; n \geq 1$

$\therefore f(n) = O(n^2)$

$2n^2 + 3n + 4 \geq 1n^2 \; , \; n \geq 1$

$\therefore f(n) = \Omega(n^2)$

$2n^2 \leq \underbrace{2n^2 + 3n + 4}_{f(n)} \leq 9n^2$

$c_1 \; g(n) \quad c_2 \; g(n)$

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

$\therefore f(n) = \Theta g(n) = \Theta(n^2)$.

② $n^2 \log n + n$

$n^2 \log n + n \leq 2n^2 \log n \;,\; n \geq 1$

$\therefore f(n) = O(n^2 \log n)$

$n^2 \log n + n \geq 1 n^2 \log n \;,\; n \geq 1$

$\therefore f(n) = \Omega(n^2 \log n)$

$1 n^2 \log n \leq n^2 \log n + n \leq 2n^2 \log n$

$\therefore f(n) = \Theta(n^2 \log n)$.

③ $f(n) = n!$

$f(n) \leq n \times n \times n \times \ldots \times n$

$\Rightarrow f(n) \leq n^n$

$\therefore f(n) = O(n^n),\; n \geq 1$

$f(n) \geq n$

$\therefore f(n) = \Omega(n), n \geq 1$

As, $O(n^n) \neq \Omega(n)$

So, Θ is not possible.

Linear search :-

Best case :- first item as search item.

$B(n) = \text{constant time} = O(1) = \Omega(1) = \Theta(1)$

Worst case :- last item

$W(n) = n = O(n) = \Omega(n) = \Theta(n)$.

Average case:-

$$\frac{1}{n} + \frac{2}{n} + \frac{3}{n} + \cdots + \frac{n}{n}$$

$$A(n) = \frac{1+2+3+\cdots+n}{n}$$

$$= \frac{\frac{n(n+1)}{2}}{n}$$

$$= \frac{n+1}{2}$$

$$= \Theta(n)$$

$$= \Omega(n)$$

$$= \Theta(n)$$

L.S.

2	5	1	8	6	10

6th class

```
test-func(n)
1. if n>0
2.     print n
3.     test-func(n-1)
```

Reccurance Relation :- [For finding complexity of recursive function]

→ Reccurance tree method

→ Substitution method

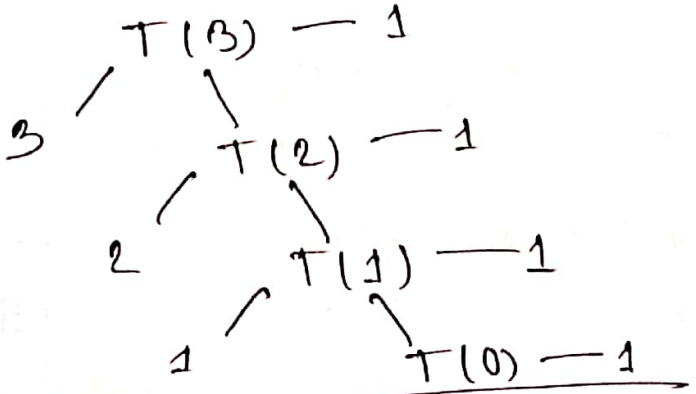
Reccurance tree method :-

1. if n>0 ——→ 1
2. print n ——→ 1
3. test-func(n-1) ——→ T(n-1)

$$T(n) = T(n-1) + 2$$

Reccurance Relation :-

$$T(n) = \begin{cases} 1, & n=0 \\ T(n-1)+1, & n>0 \end{cases}$$



$$T(n) = 3 + 1$$

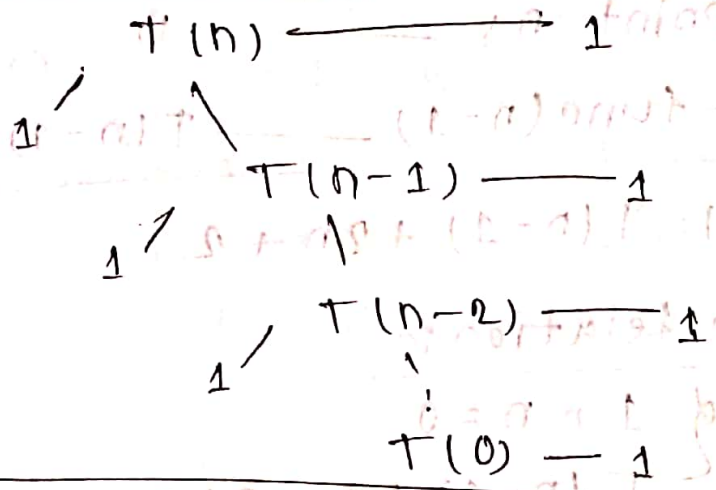
$$\Rightarrow T(n) = n + 1 \quad \therefore \Theta(n)$$

For the last call,

$n - k = 0$

$n = k$

Recurrance tree :-



$T(n) = n + 1 = \theta(n)$

Substitution method :-

$T(n) = T(n-1) + 1$; [$T(n-1) = T(n-2) + 1$]

$= T(n-2) + 1 + 1$; [$T(n-2) = T(n-3) + 1$]

$= T(n-3) + 1 + 1 + 1$;

$\vdots$

$= T(n-k) + k$ ——— (1)

$T(0)$

for the last call,

$n - k = 0$

$\therefore n = k$

by substituting this into (1),

$T(n) = T(n-n) + n$

$= T(0) + n$

$= n + 1$

$= \theta(n) / O(n)$.

T(n) - test-func(n) :-

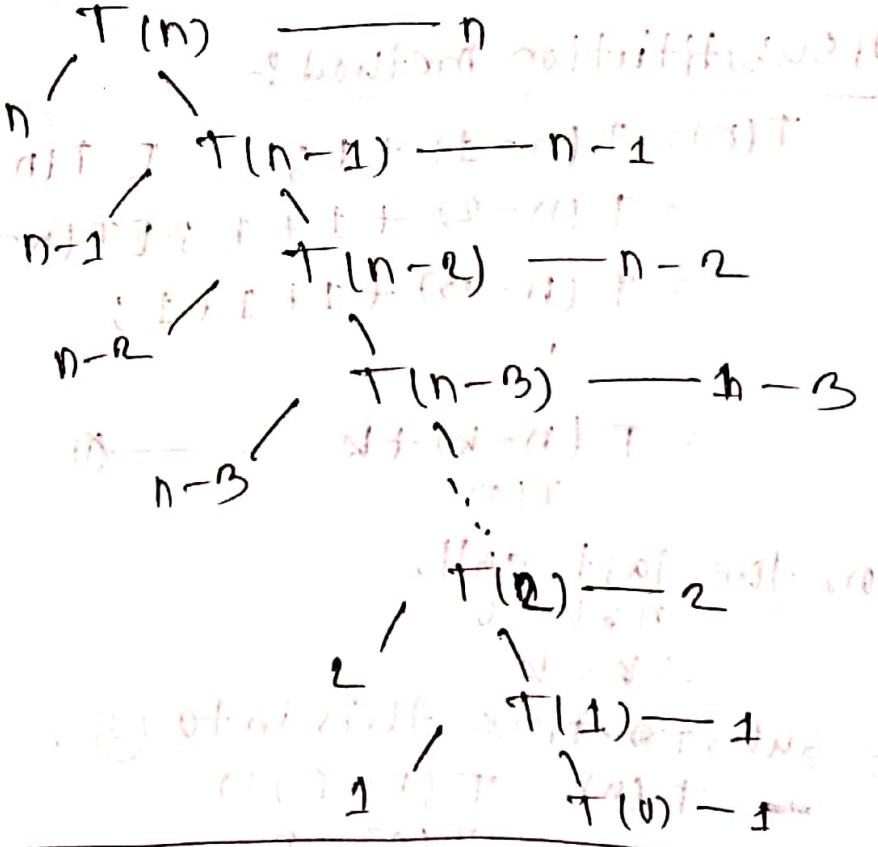
```
1. if n>0 ———————— 1
2.    for i=0 to (n-1); ——— n+1
3.        Print n; ———————— n
4.    test-func(n-1) ——— T(n-1)
```

$$T(n) = T(n-1) + 2n + 2$$

Recurrance Relation :-

$$T(n) = \begin{cases} 1, & n = 0 \\ T(n-1) + n, & n > 0 \end{cases}$$

R. Tree :-



$$T(n) = 1 + 2 + 3 + \cdots + (n-1) + n$$
$$= 1 + \frac{n(n+1)}{2}$$

$= 1 + \frac{n^2+n}{2}$

$\therefore \Theta(n^2)$

Back-Substitution method :-

$T(n) = T(n-1) + n$

$= [T(n-2) + n-1] + n$

$= T(n-2) + (n-1) + n$

$= T(n-3) + (n-2) + (n-1) + n$

$[T(n-2) = T(n-3) + (n-2)]$

$\vdots$

$= T(n-k) + n - (k-1)$

$T(n) = T(n-k) + (n-(k-1)) + (n(k-2)) + \cdots + (n-1) + n$ —①

for last substitution,

$n - k = 0$

$\therefore n = k$

Putting value in ①,

$T(n) = T(n-n) + (n-n+1) + (n-n+2) - \cdots + (n-1) + n$

$= T(0) + 1 + 2 + 3 + \cdots + (n-1) + n$

$= 1 + \frac{n(n+1)}{2}$

$= \Theta(n^2)$,

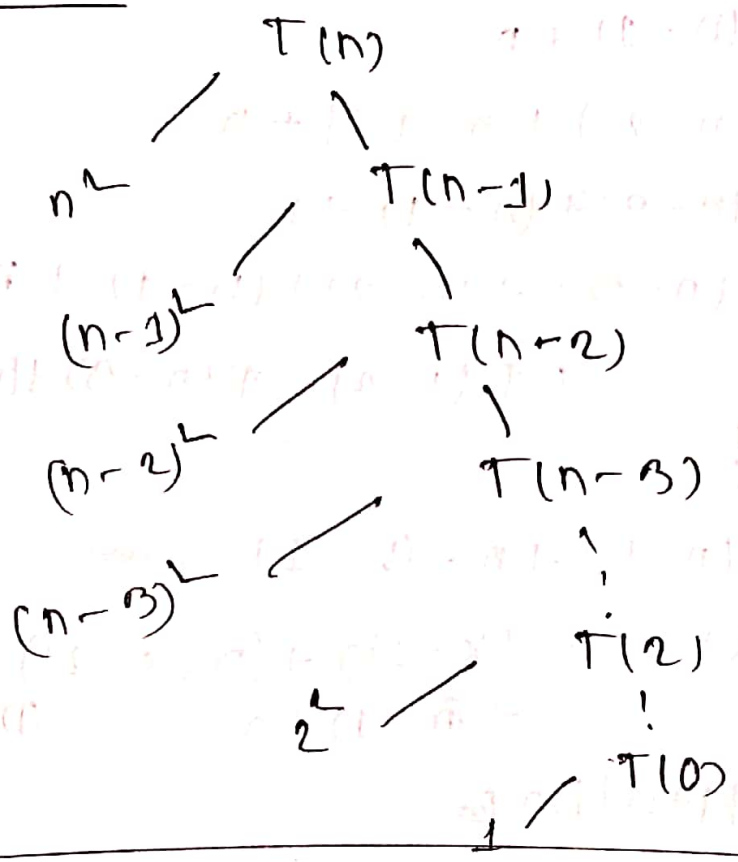
7th class

~~Ineq~~ :-

R.R: $T(n) = 1 \quad , \quad n = 0$

$T(n-1) + n^2 \quad , \quad n > 0$

R.T :



$$T(n) = 1 + (n-1)^2 + (n-2)^2 + \cdots + (n-k)^2 + n^2$$

$$= 1 + \frac{n^2(n+1)^2}{2} + n^2$$

$$= 1 + n^3$$

$$= O(n^3)$$

Substitute :-

$$T(n) = T(n-1) + n^2$$

$$= T(n-2) + (n-1)^2 + n^2$$

$$= T(n-3) + (n-2)^2 + (n-1)^2 + n^2$$

$$\vdots$$

$$= T(n-k) + (n-k+1)^2 + (n-k-2)^2 + \cdots + (n-1)^2 + n^2 \quad \text{—①}$$

For kth step,

$$n - k = 0$$

$$\therefore n = k$$

from equ —①

$$T(n) = T(0) + 1 + 2^2 + 3^2 + \cdots + (n-1)^2 + n^2$$

$$= 1 + n^2 + (n-1)^2 + n^2$$

$$= 1 + n^3$$

$$= O(n^3)$$

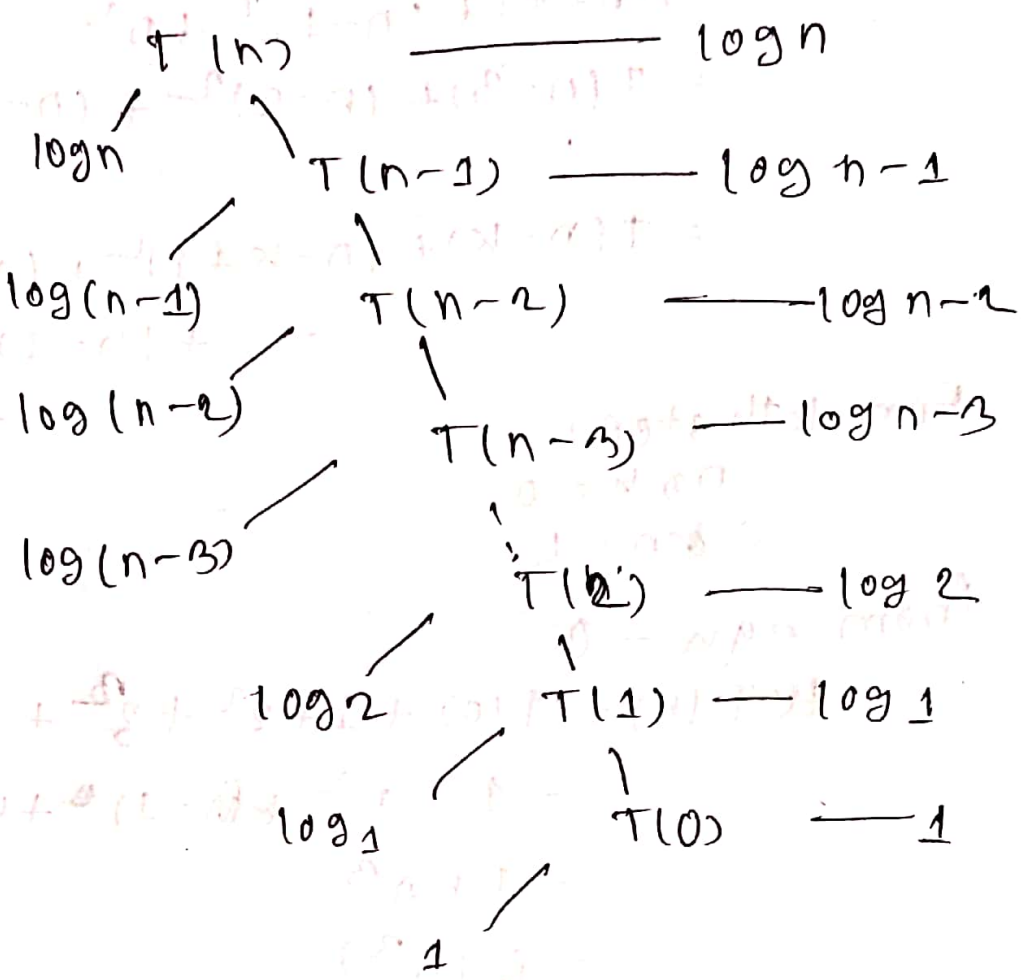
Test - func (n) :-

```
1. if n>0
2.   for (i=1; i<n; i*=2) {
3.       print i;
4.   test-func (n-1)
```

$$T(n) = T(n-1) + \log n + 1$$

R.R :- $T(n) = 1, \quad n = 0$

$T(n-1) + \log n, \quad n > 0$



$T(n) = 1 + \log 1 + \log 2 + \log 3 + \cdots + \log n$

$= 1 + 0 + 1 + \log 2 + \log 3 + \cdots + \log n$

$= 1 + \log [1 \times 2 \times 3 \times \cdots \times n]$

$= 1 + \log n!$

$= 1 + \log (n^n)$

$= 1 + n \log n$

$= O(n \log n)$.

Substitution:

$$T(n) = T(n-1) + \log n$$

$$= T(n-2) + \log(n-1) + \log n$$

$$= T(n-3) + \log(n-2) + \log(n-1) + \log n$$

$$= T(n-k) + \log(n-k+1) + \log(n-k+2) + \cdots + \log n$$

$n - k = 0$

$n = k$

$$= T(0) + \log 1 + \log 2 + \log 3 + \cdots + \log n$$

$$= 1 + \log[1 \times 2 \times 3 \times \cdots \times n]$$

$$= 1 + \log n!$$

$$= 1 + \log n^n$$

$$= 1 + n \log n$$

$$= O(n \log n).$$

$\log(1 \times 2 \times \cdots \times n) \leq \log(n \times n \times n \times \cdots \times n)$

$\leq \log n^n$

$\Rightarrow \log n! \leq n \log n$

$\therefore f(n) = O(n \log n)$

Shortcut:-

$T(n) = T(n-1) + 1$ ——— $O(n)$

$T(n) = T(n-1) + n$ ——— $O(n^2)$

$T(n) = T(n+1) + n^2$ ——— $O(n^3)$

$T(n) = T(n+1) + n^4$ ——— $O(n^5)$

$T(n) = T(n-1) + \log n$ ⟶ $O(n \log n)$

$T(n) = T(n-1) + n \log n$ ⟶ $O(n^2 \log n)$.

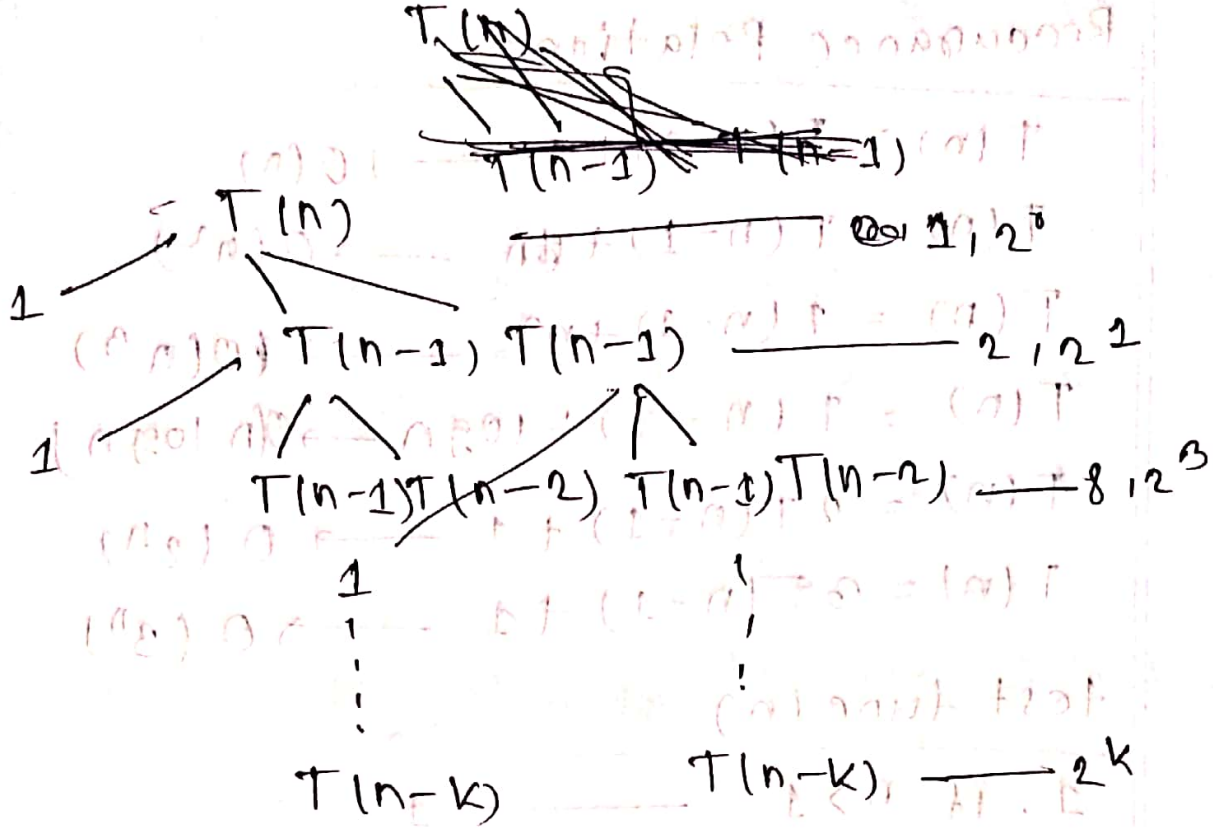
Test-func(n)

```
1. if n>0 ————————→ 1
2.    print n; ————————1
3.    test func(n-1) ——————T(n-1)
4.    test func(n-1) ——————→T(n-1)
```

$T(n) = 2T(n-1) + 2$

$T(n) = 1 \quad , \quad n = 0$

$2T(n-1) + 1 \quad , \quad n > 0$



$$T(n) = 1 + 2 + 2^2 + 2^3 + \cdots + 2^k$$

$a + ar + ar^2 + \cdots + ar^k$

$a = 1, r = 2$

$$= \frac{a(r^{(k+1)} - 1)}{r - 1}$$

$$= \frac{1(2^{(k+1)} - 1)}{2 - 1}$$

$$= 2^{k+1} - 1$$

$$= 2^{n+1} - 1$$

$$= O(2^n)$$

8th class

Reccurance Relation:-

$T(n) = T(n-1) + 1 \longrightarrow O(n)$

$T(n) = T(n-1) + n \longrightarrow O(n^2)$

$T(n) = T(n-1) + n^2 \longrightarrow O(n^3)$

$T(n) = T(n-1) + \log n \longrightarrow O(n \log n)$

$T(n) = 2T(n-1) + 1 \longrightarrow O(2^n)$

$T(n) = 3T(n-1) + 1 \longrightarrow O(3^n)$

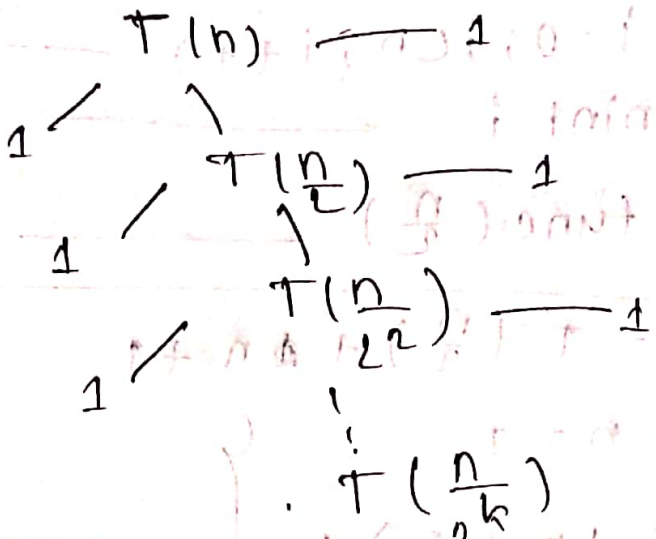
test fune(n)

1. if $n > 1$ ——— 1
2. print n ——— 1
3. test-fune $(\frac{n}{2})$ ——— $\frac{n}{2} T(\frac{n}{2})$

$T(n) = T(\frac{n}{2}) + 2$

$$T(n) = \begin{cases} 1, & n = 1 \\ T(\frac{n}{2}) + 1, & n > 1 \end{cases}$$

R.T :-



~~$T(n) = T\left(\frac{n}{2^k}\right) + k$~~

For last call,

$\frac{n}{2^k} = 1$, $k = \log n$

$T(n) = \log n$

$O(\log n)$

Substitution :-

$T(n) = T\left(\frac{n}{2}\right) + 1$

$= \left[T\left(\frac{n}{2^2}\right) + 1\right] + 1$ $\left[T\left(\frac{n}{2}\right) = T\left(\frac{n}{2^2}\right) + 1\right]$

$= T\left(\frac{n}{2^3}\right) + 2$

$= T\left(\frac{n}{2^4}\right) + 3$

$= T\left(\frac{n}{2^k}\right) + k$ ——— (1)

$\frac{n}{2^k} = 1$, $k = \log n$

$= T(1) + \log n$

$= 1 + \log n$

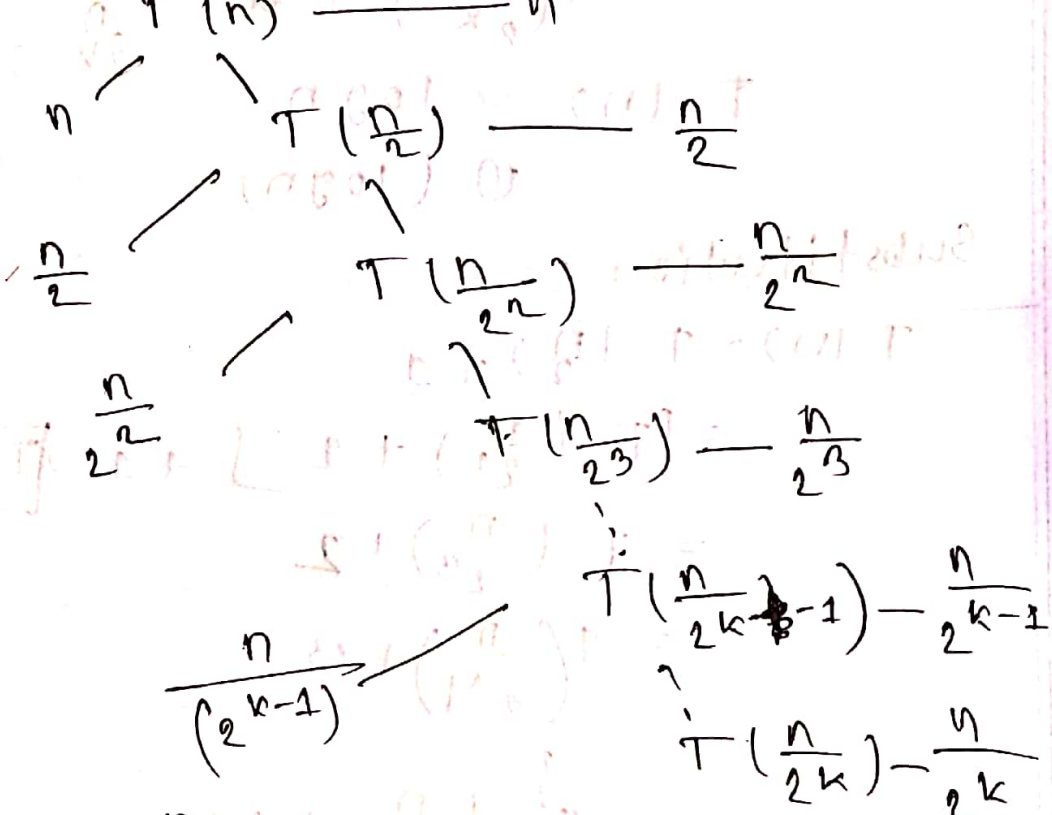
$= O(\log n)$

1. if n > 1
2. for (i=0; i<n; i++) —— n
3. Print i —— 1
4. test-func($\frac{n}{2}$) —— $T(\frac{n}{2})$

$T(n) = T(\frac{n}{2}) + n + 1$

$$T(n) = \begin{cases} 1, & n = 1 \\ T(\frac{n}{2}) + n, & n > 1 \end{cases}$$

R.T:



$$T(n) = n + \frac{n}{2} + \frac{n}{2^2} + \cdots + \frac{n}{2^k}$$

$$= n + n\underbrace{\left(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^k}\right)}_{\leq 1}$$

$$\leq n + n$$

$$= 2n = O(n)$$

substitute :-

$$T(n) = T\left(\frac{n}{2}\right) + n$$

$$= \left[T\left(\frac{n}{2}\right) + \frac{n}{2}\right] + n$$

$$= \left[T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n$$

$$= \left[T\left(\frac{n}{2^3}\right) + \frac{n}{2}\right] + n$$

$$\vdots$$

$$= \left[T\left(\frac{n}{2^k}\right) + \frac{n}{2^{k-1}}\right] + \cdots + \frac{n}{2} + n$$

$$= 1 + n + n\underbrace{\left(\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^{k-1}}\right)}_{\leq 1}$$

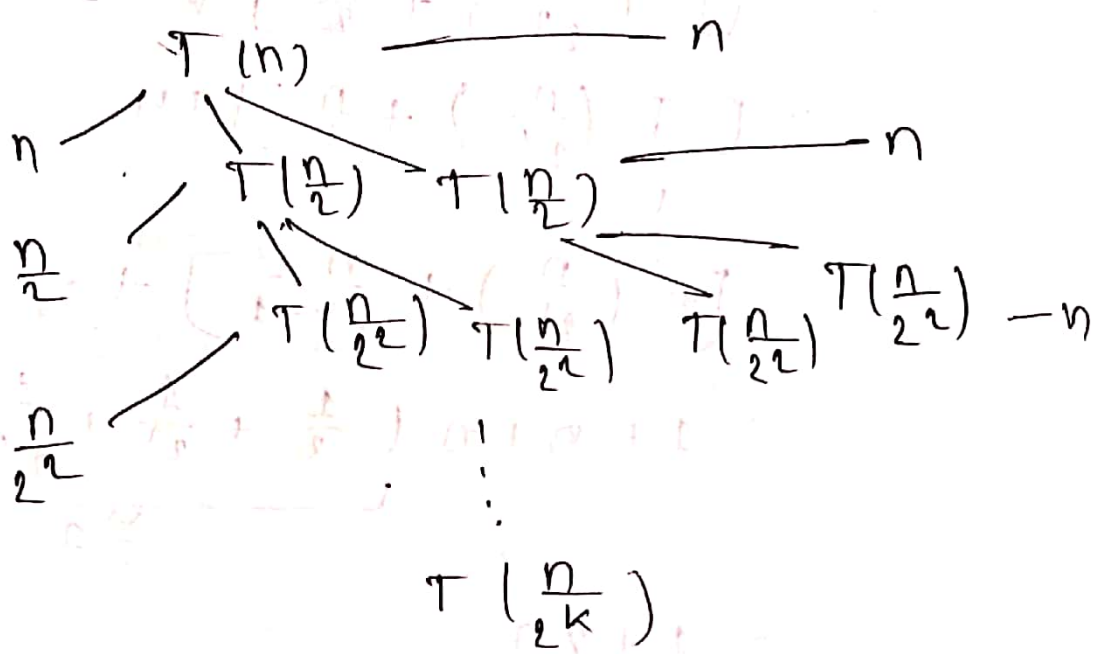
$$= 1 + 2n$$

$$= O(n).$$

Again,

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & n > 1 \end{cases}$$

R.T:-



$$T(n) = n \times k = n \log n$$
$$= O(n \log n).$$

Substitution:-

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$
$$= 2\left[2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n$$
$$= 2^2 T\left(\frac{n}{2^2}\right) + n + n$$
$$= 2^2 \left[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right] + 2n,$$
$$\left[T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + \frac{n}{2}\right]$$

$$= 2^3\left[T\left(\frac{n}{2^3}\right)\right] + 3n$$

$$\vdots$$

$$= 2^k T\left(\frac{n}{2^k}\right) + kn \quad \text{——} (1)$$

$$\frac{n}{2^k} = 1$$

$$k = \log n$$

$$\therefore T(n) = n\,T(1) + n\log n$$

$$= n + n\log n$$

$$= O(n\log n)$$

9th class

Analysis of binary search :-

2	4	5	9	10	20	21	25
1	-	-	-	-	-	-	8

```
Binary search (n, l, u, key)
{
    if (l ≤ u)
        mid = (l+u)/2 ———————— 1
        if (A[mid] == key)
            return ~~key~~ mid ———— 1
        else if (A[mid] > key)
            Binary search (A, l, mid-1, key)
        else                                  } T(n/2)
            Binary search (A, mid+1, u, key)
    else
        return -1; ———————— 1
}
```

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$$T(n) = \begin{cases} 1 & , n = 1 \\ T\left(\frac{n}{2}\right) + 1 & , n > 1 \end{cases}$$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$$= O(\log_2 n)$$

$\frac{n}{2}$ — n

$\frac{n}{2^2}$ — n

$\frac{n}{2^3}$ — n

$\frac{n}{2^k}$ — n

$\frac{n}{2^k} = 1$ ∴ k = log n

Analysis of Merge sort :-

Merge sort (A, l, u) :-

```
if (l < u)
    mid = (l+u)/2          ——— 1
    Merge-sort(l, mid)     ——— T(n/2)
    Merge-sort(mid+1, u)   ——— T(n/2)
    Merge(A, l, mid, u)    ——— n
```

$$T(n) = 2T\left(\frac{n}{2}\right) + n + 1$$

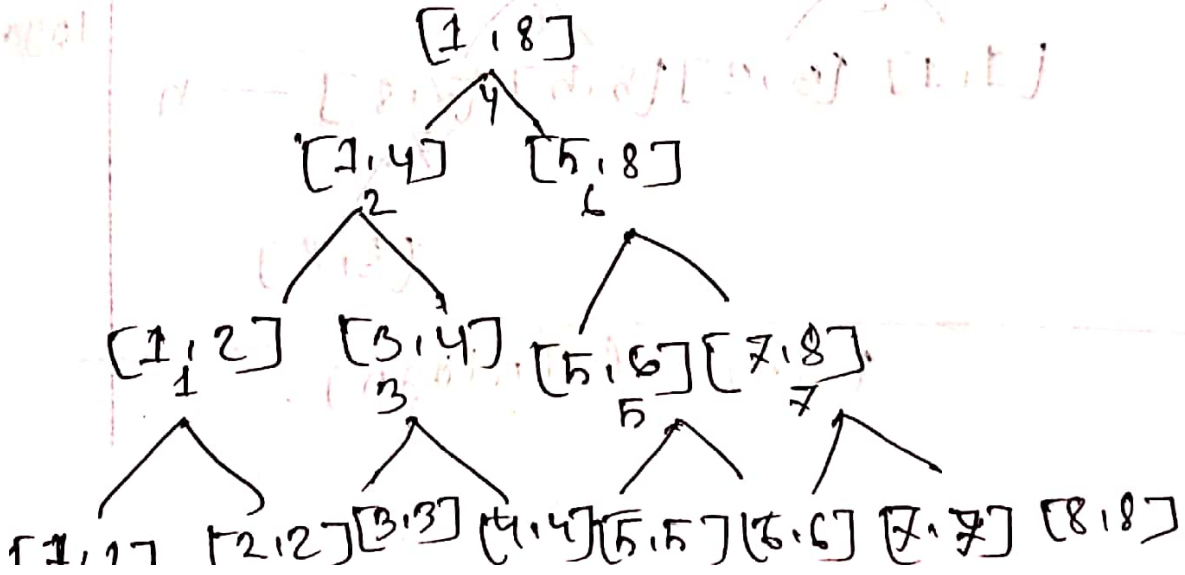
R.R :-

$T(n) = 1, \quad n = 1$ ——— $O(1)$

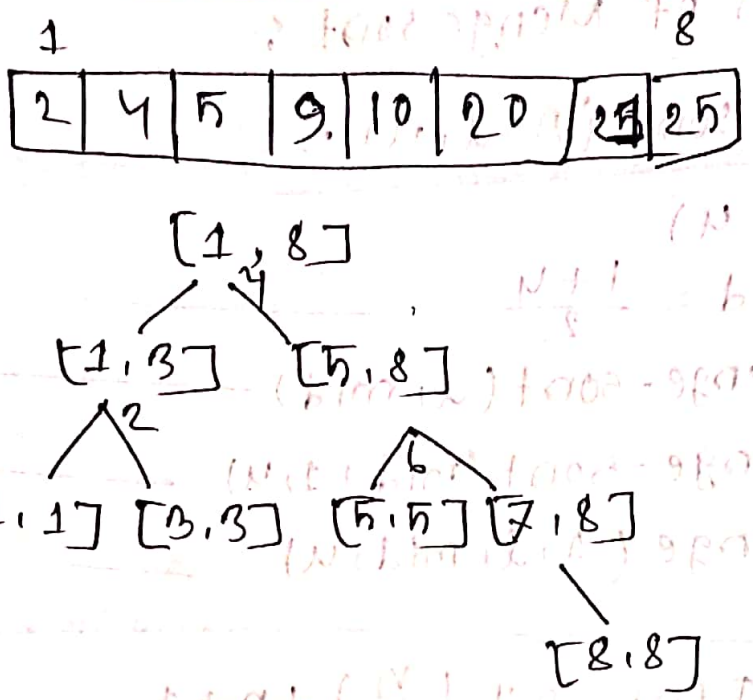
$2T\left(\frac{n}{2}\right) + n, \quad n > 1$ ——— $O(n \log n)$

M.S :

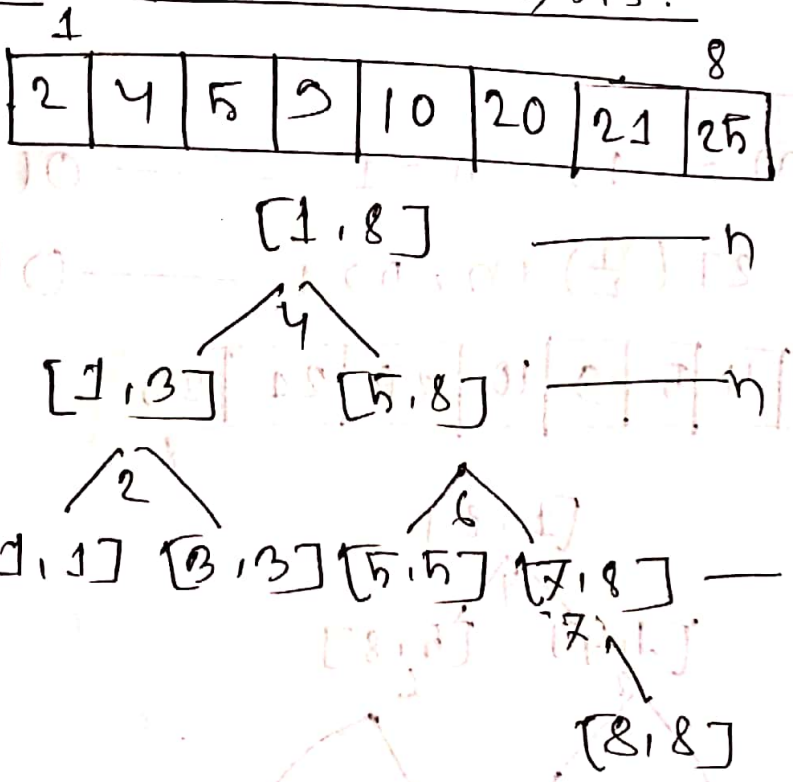
2	4	5	9	10	20	21	25
1							8



B.S :-



Q.S :- Quicksort Analysis :-



$T(n) = O(n \log n)$.

T(n) Quicksort (A, l, u)

```
if(l < u)
    pnt = partition(A, l, u, pivot)      ——— n
    Quicksort(A, l, pnt-1)               ——— T(n/2)
    Quicksort(A, pnt+1, n)               ——— T(n/2)
```

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Rec Rel :-

$$T(n) = \begin{cases} 1 & , n = 1 \quad \text{——— } O(1) \\ 2T\left(\frac{n}{2}\right) + n & , n > 1 \quad \text{——— } O(n \log n) \end{cases}$$

Best case :- (when ~~search~~ elements are already sorted)

[1, 8] ——— n

/

[1, 7] ——— n-1

/

[1, 6] ——— n-2

⋮

[1, 1] ——— 1

$$T(n) = 1 + 2 + \cdots + n$$

$$= \frac{n(n+1)}{2}$$

$$= O(n^2)$$